Name: _____________________________ Student No: _____________________________

Department: ___________________________ Tutorial Section: ___________________________

This is a closed book exam. You should put everything off your desk except for stationaries and your student ID. You won’t need a calculator.

You should use the back of the pages for rough work and, if necessary, for your final answers. There is one blank page at the end in case you need more space. Write legibly and don’t use pencils for your final answers.

Be concise with your answers. Marks might be deducted for incorrect redundant statements. When describing algorithms, you can use either natural language or pseudo code as long as your descriptions are clear.

<table>
<thead>
<tr>
<th>Question</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
</tbody>
</table>
1. (5 Points)
Consider two complex numbers $a + bi$ and $c + di$ where $i = \sqrt{-1}$. How to compute their product with only 3 numerical multiplications? (Note that there is no requirement on the number of additions and subtractions.)

Solution:

The product of the two complex numbers $a+bi$ and $c+di$ can be computed as follows:

$$(a + bi)(c + di) = \alpha + (\beta - \alpha - \gamma)i - \gamma$$

where $\alpha = ac$, $\beta = (a + b)(c + d)$ and $\gamma = bd$ which require only 3 numerical multiplications.

2. (5 Points)
Give an example of a graph for which Approx-Vertex-Cover always yields a suboptimal solution.

Solution:

An example of a graph for which Approx-Vertex-Cover always yields a suboptimal solution is as follows.

Other possible solutions:
3. (10 Points)
What are the operations in the disjoint set UNION-FIND data structure? How are they implemented? (Describe the basic implementation. There is no need mention path compression.) What is the time complexity of each operation?

Solution:

The UNION-FIND data structure supports the following 3 operations. [ 3 points ]

(1 point for each operation)

(1) Create-Set(x): Create a set containing a single item x.
(2) Find-Set(x): Find the set that contains x, and
(3) Union(x,y): Merge the set containing x, and another set containing y to a single set.

Each of these operations are implemented as follows. [ 4 points ]

(1 1 point for Create-Set(x), 1 point for Find-Set(x) and 2 points for Union(x,y))

(1) Create-Set(x):

Create-Set(x){
    x->parent = x;
}

(2) Find-Set(x): We simply trace the parent pointer until we reach the root, then return the root element.

Find-Set(x){
    while(x! = x->parent)
        x = x->parent;
    return x;
}

(3) Union(x,y): We always make the root of taller tree the parent of shorter tree. The root of every tree also keeps track of the height of the tree. In case two trees has the same height, we choose the root of the first tree to point to the root of second. And the tree height is increased by 1.

Union(x,y){
    a = Find-Set(x);
    b = Find-Set(y);
    if(a.height <= b.height){
        if(a.height == height)
            b.height++;
        a->parent = b;
    }
    else
        b->parent = a;
}

[ 3 points ]
(1 1 point for each operation)

For n items, the running time of Create-Set is O(1), Find-Set is O(logn), and Union is O(logn) respectively.
4. (10 Points) 
What is the optimal way to compute $A_1A_2A_3A_4A_5$, where the dimensions of the matrices are: $A_1$: $10 \times 20$, $A_2$: $20 \times 2$, $A_3$: $2 \times 40$, $A_4$: $40 \times 5$, $A_5$: $5 \times 30$? Show the table for the bottom-up computation.

Solution:

**Bottom-up computation of $m[i, j]$**: [7.5 points]

<table>
<thead>
<tr>
<th>$m[i, j]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>22</td>
</tr>
</tbody>
</table>

$\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
2 & 3 & 4 & 5 & 6 \\
3 & 4 & 5 & 6 & 7 \\
4 & 5 & 6 & 7 & 8 \\
5 & 6 & 7 & 8 & 9 \\
\end{array}$

<table>
<thead>
<tr>
<th>$s[i, j]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$p_0$</th>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$p_3$</th>
<th>$p_4$</th>
<th>$p_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

[ | 2.5 points for showing the optimal way of parenthesizing the product ]

Least number of multiplications needed to compute the product = 1700.

From the entries of $s[i, j]$, we see that

- The last split of $A_{1..5}$ should be made at the position $s[1..5] = 2$. So we have $(A_1A_2)(A_3A_4A_5)$.
- To determine where to split the three matrices $A_3A_4A_5$, we look at the entry $s[3, 5] = 4$. So the split should be made at the position 4. Thus, we have $(A_1A_2)((A_3A_4)A_5)$.

Optimal way of parenthesizing the product: $(A_1A_2)((A_3A_4)A_5)$.

(FOR REFERENCE ONLY)

**Computation**:

**Step 0**: Initialize all the base cases:

$m[1, 1] = 0, m[2, 2] = 0, m[3, 3] = 0, m[4, 4] = 0, m[5, 5] = 0$

**Step 1**: Find the solution of subproblems with 2 matrices

$m[1, 2] = m[1, 1] + m[2, 2] + 10 \cdot 20 \cdot 2 = 0 + 0 + 400 = 400$, $s[1, 2] = 1$

$m[2, 3] = m[2, 2] + m[3, 3] + 20 \cdot 2 \cdot 40 = 0 + 0 + 1600 = 1600$, $s[2, 3] = 2$

$m[3, 4] = m[3, 3] + m[4, 4] + 2 \cdot 40 \cdot 5 = 0 + 0 + 400 = 400$, $s[3, 4] = 3$

$m[4, 5] = m[4, 4] + m[5, 5] + 40 \cdot 5 \cdot 30 = 0 + 0 + 6000 = 6000$, $s[4, 5] = 4$
**Step 2:** Find the solution of subproblems with 3 matrices

\[
m[1,3] = \min \left\{ \begin{array}{l}
m[1,1] + m[2,3] + 10 \cdot 20 \cdot 40, \\
m[1,2] + m[3,3] + 10 \cdot 2 \cdot 40 \\
\end{array} \right\}
\]

\[
= \min \left\{ \begin{array}{l}
0 + 1600 + 8000, \\
400 + 0 + 800 \\
\end{array} \right\}
\]

\[
= \min \left\{ 9600, 1200 \right\} = 1200
\]

\[
s[1,3] = 2
\]

\[
m[2,4] = \min \left\{ \begin{array}{l}
m[2,2] + m[3,4] + 20 \cdot 2 \cdot 5, \\
m[2,3] + m[4,4] + 20 \cdot 40 \cdot 5 \\
\end{array} \right\}
\]

\[
= \min \left\{ \begin{array}{l}
0 + 400 + 200, \\
1600 + 0 + 4000 \\
\end{array} \right\}
\]

\[
= \min \left\{ 600, 5600 \right\}
\]

\[
s[2,4] = 2
\]

\[
m[3,5] = \min \left\{ \begin{array}{l}
m[3,3] + m[4,5] + 2 \cdot 40 \cdot 30, \\
m[3,4] + m[5,5] + 2 \cdot 5 \cdot 30 \\
\end{array} \right\}
\]

\[
= \min \left\{ \begin{array}{l}
0 + 6000 + 2400, \\
400 + 0 + 300 \\
\end{array} \right\}
\]

\[
= \min \left\{ 8400, 700 \right\} = 700
\]

\[
s[3,5] = 4
\]

**Step 3:** Find the solution of subproblems with 4 matrices

\[
m[1,4] = \min \left\{ \begin{array}{l}
m[1,1] + m[2,4] + 10 \cdot 20 \cdot 5, \\
m[1,2] + m[3,4] + 10 \cdot 2 \cdot 5, \\
m[1,3] + m[4,4] + 10 \cdot 40 \cdot 5 \\
\end{array} \right\}
\]

\[
= \min \left\{ \begin{array}{l}
0 + 600 + 1000, \\
400 + 400 + 100, \\
1200 + 0 + 2000 \\
\end{array} \right\}
\]

\[
= \min \left\{ 1600, 900, 3200 \right\} = 900
\]

\[
s[1,4] = 2
\]

\[
m[2,5] = \min \left\{ \begin{array}{l}
m[2,2] + m[3,5] + 20 \cdot 2 \cdot 30, \\
m[2,3] + m[4,5] + 20 \cdot 40 \cdot 30, \\
m[2,4] + m[5,5] + 20 \cdot 5 \cdot 30 \\
\end{array} \right\}
\]

\[
= \min \left\{ \begin{array}{l}
0 + 700 + 1200, \\
1600 + 6000 + 24000, \\
600 + 0 + 3000 \\
\end{array} \right\}
\]

\[
= \min \left\{ 1900, 31600, 3600 \right\} = 1900
\]

\[
s[2,5] = 2
\]
**Step 4:** Find the solution of subproblems with 5 matrices

\[
m[1, 5] = \min \left\{ \begin{array}{l}
m[1, 1] + m[2, 5] + 10 \cdot 20 \cdot 30, \\
m[1, 2] + m[3, 5] + 10 \cdot 2 \cdot 30, \\
m[1, 3] + m[4, 5] + 10 \cdot 40 \cdot 30, \\
m[1, 4] + m[5, 5] + 10 \cdot 5 \cdot 30 \\
\end{array} \right. \\
= \min \left\{ \begin{array}{l}
0 + 1900 + 6000, \\
400 + 700 + 600, \\
1200 + 6000 + 12000, \\
900 + 0 + 1500 \\
\end{array} \right. \\
= \min \{ 7900, 1700, 19200, 2400 \} = 1700 \\
s[1, 5] = 2
\]
5. (20 Points)  
Yuckdonald’s is considering opening a series of restaurants along Quaint Valley Highway (QVH). The $n$ possible locations are along a straight line, and the distances of these locations from the start of QVH are, in miles and in increasing order: $m_1, m_2, \ldots, m_n$. The constraints are as follows:

1. At each location, Yuckdonald’s may open at most one restaurant. The expected profit from opening a restaurant at location $i$ is $p_i$, where $p_i > 0$ and $i = 1, 2, \ldots, n$.
2. Any two restaurants should be at least $k$ miles apart, where $k$ is a positive integer.

Give a dynamic programming algorithm that determines the locations to open restaurants which maximizes the total expected profit.

Solution:

**Subproblems Definition:** [3 points]

We define $T[i]$ to be the total profits from the best legitimate configuration using locations $1, 2, \ldots, i$ only. We also store $R[i]$ which is 1 if there is a restaurant at location $i$ and 0 otherwise.

**Recursive Formulation:** [10 points]

**Case 1: Base case** (↓ 2 points)

If $i = 0$, then there is no location available to choose from to open a restaurant. So $T[0] = 0$.

**Case 2: General case**

If $i > 0$, then we have two options. They are

1. Do not open a restaurant at location $i$ (↓ 3 points)

   If we choose to not open a restaurant at location $i$, then the optimal value will come about by considering how to obtain total profits from the best legitimate configuration using the remaining location $1, 2, \ldots, i - 1$. This is just $T[i - 1]$.

2. Open a restaurant at location $i$ (↓ 3 points)

   If we open a restaurant at location $i$, then we gain the expected profit $p_i$. As we want to build a restaurant at location $i$, then the closest location to build another restaurant should be at $c_i$, where $c_i$ denote the maximum $j$ which $m_j \leq m_i - k$. To obtain a maximum profit, we need to obtain the maximum profits from the remaining locations $1, 2, \ldots, c_i$. This is just $p_i + T[c_i]$.

Since these are the only two possibilities, we can see that we have the following rule for constructing table $T$:

$$
T[i] = \begin{cases} 
0 & \text{if } i = 0 \\
\max\{T[i - 1], p_i + T[c_i]\} & \text{if } i > 0 
\end{cases}
$$

(↓ 2 points for recording restaurants position)

If $T[i] = T[i - 1]$, then $R[i] = 0$ and $R[i] = 1$ otherwise.

Note: We can compute $c_i = \max\{j : m_j \leq m_i - k\}$ for every $i$. Note that for some values of $i$ and $c_i$ may not exist in which case, we assume that $c_i = 0$. 


**Bottom-up Computation:** [4 points]

We compute and save $T[i]$ in such an order that:
When it is time to compute $T[i]$, the values of $T[i-1]$ and $T[c_i]$ are available. So we will fill the table in an order of increasing $i$.

By following the ideas described above, we have the algorithm presented in pseudocode as follows.

```
// Algorithm to compute $c_i$ for every $i$
Compute_ci(m1,...,mn, k){
    Initialize $m_i' = m_i - k$ for all $i$
    Merge the sorted arrays $\{m_1,...,m_n\}$ and $\{m_1',...,m_n'\}$ into $A$
        (when there is a tie in comparing $m_i$ and $m_j'$ for some $i$ and $j$, put $m_j'$ into $A$ first)
    $t = 0$;
    for (j=1 to 2n){
        if (A[j] == $m_i'$ for some $i$)
            set $c_i = t$;
        else
            $t++$;
    }
}
```

```
// Algorithm to find optimal profit and locations to open restaurants
Find_Optimal_Profit_And_Pos(m1...mn, p1...pn, c1...cn){
    $T[0] = 0$;
    for (i=1 to n){
        Not_Open_At_I = $T[i-1]$;
        Open_At_I = $p_i + T[c_i]$;
        if(Not_Open_At_I > Open_At_I){
            $T[i] = Not_Open_At_I$;
            $R[i] = 0$;
        }
        else{
            $T[i] = Open_At_I$;
            $R[i] = 1$;
        }
    }
    return $T[n]$ and $R$;
}
```

```
// Algorithm to report optimal locations to open restaurants
Report_Optimal_Locations(R, c1..cn){
    $j = n$;
    $S$ = empty set;
    while ($j >= 1$){
        if ($R[j] = 1$){
            Insert $m_j$ into $S$;
            $j = c_j$;
        }
        else
            $j = j - 1$;
    }
    return $S$;
}
```
**Running Time Analysis:** [ 3 points ]

- The $\text{Compute}_c i$ takes $O(n)$ time to compute $c_i$ for every $i$.
- The $\text{Find\ Optimal\ Profit\ And\ Pos}$ takes $O(n)$ time to compute $T$ and $R$.
- The $\text{Report\ Optimal\ Locations}(R, c_1..c_n)$ takes $O(n)$ time to report the optimal locations for opening restaurants along Quaint Valley Highway.

Therefore, the overall running time is $O(n)$. 
6. (20 Points)
Consider the following problem. The input consists of \( n \) skiers with heights \( p_1, \ldots, p_n \), and \( n \) skies with heights \( s_1, \ldots, s_n \). The problem is to assign each skier a ski to minimize the average squared difference between the height of a skier and his/her assigned ski. That is, if the \( i \)-th skier is given the \( \alpha(i) \)-th ski, then you want to minimize:

\[
\frac{1}{n} \sum_{i=1}^{n} (p_i - s_{\alpha(i)})^2
\]

Give a greedy algorithm to do the ski assignment so as to minimize the average difference. Prove the correctness of your algorithm.

Solution:

**Greedy algorithm:** [5 points]

The greedy algorithm is presented as follows:

We will give the shortest skier the shortest ski, give the second shortest skier the second shortest ski, give the third shortest skier the third shortest ski and so on.

**Correctness proof:** [15 points]

Assume that the skiers and skis are numbered in increasing order by height.

(↓ 3 points)

Let the greedy solution be \( G = \{(p_1, s_1), \ldots, (p_n, s_n)\} \) and consider any optimal solution \( O = \{(p_1, s'_1), \ldots, (p_n, s'_n)\} \). Let's start with \( p_1 \), compare \( G \) and \( O \). Let \( p_i \) be the first person who is assigned different skis in \( G \) than in \( O \). Let \( s_\beta \) be the skis assigned to \( p_i \) (where \( i + 1 \leq \beta \leq n \)) in \( O \) and suppose \( s_i \) is assigned to \( p_\gamma \) (where \( i + 1 \leq \gamma \leq n \)).

(↓ 1 point)

Create solution \( O' \) by switching the ski assignments of \( p_i \) and \( p_\gamma \).

(↓ 4 points)

By the definition of greedy algorithm, \( s_i \leq s_\beta \). The average squared difference of \( O' \) is given by

\[
\text{AverageSqrDiff}(O') = \text{AverageSqrDiff}(O) - \frac{1}{n} (p_i - s_\beta)^2 - \frac{1}{n} (p_\gamma - s_i)^2 + \frac{1}{n} (p_i - s_i)^2 + \frac{1}{n} (p_\gamma - s_\beta)^2
\]

\[= \text{AverageSqrDiff}(O) - \frac{1}{n} ((p_i - s_\beta)^2 + (p_\gamma - s_i)^2 - (p_i - s_i)^2 - (p_\gamma - s_\beta)^2)\]

(↓ 3 points)

Now, we need to show that \((p_i - s_\beta)^2 + (p_\gamma - s_i)^2 - (p_i - s_i)^2 - (p_\gamma - s_\beta)^2 \geq 0\).

\[
= (p_i - s_\beta)^2 + (p_\gamma - s_i)^2 - (p_i - s_i)^2 - (p_\gamma - s_\beta)^2
\]

\[= -2p_is_\beta - 2p_\gamma s_i + 2p_is_i + 2p_\gamma s_\beta
\]

\[= 2(p_i(s_i - s_\beta) - 2p_\gamma(s_i - s_\beta)
\]

\[= 2(p_i - p_\gamma)(s_i - s_\beta)
\]

\[\geq 0\] Since \( p_i \leq p_\gamma \) and \( s_i \leq s_\beta \)

10
The average squared difference of the new assignment $O'$ is smaller than or equal to that of the original assignment $O$. Thus the new assignment $O'$ is also optimal.

By repeating this process, we will eventually convert $O$ into $G$, without increasing the average squared difference. Therefore $G$ is also optimal.
7. (15 Points)
Let $m$ and $n$ be two positive integers. Consider the following brute-force algorithm for finding the greatest common divider of $m$ and $n$:

\[
\text{GCD}(m, n)
\]

- let $r = 1$.
- for $(k = 1; k \leq \min(m, n); k++)$
  - if $k$ divides both $m$ and $n$, $r = k$.
- return $r$.

Give a reasonable measure for the size of the input to GCD, and analyze the complexity of GCD in terms of this input size. (You can assume that $m \leq n \leq am^b$ for some positive constants $a$ and $b$.)

**Solution:**

(↓ 4 points for giving reasonable measure for the input size)

A reasonable measure for the size of the input to GCD is $I = \log_2 m + \log_2 n$.

(↓ 2 points for representing $m$ in terms of $I$)

To represent $m$ in terms of $I$, we have

\[
\begin{align*}
I &= \log_2 m + \log_2 n \\
&\geq 2\log_2 m \\
m &\leq 2^I
\end{align*}
\]

(↓ 9 points for analyzing complexity of the algorithm)

The complexity of the brute force GCD algorithm is

\[
\begin{align*}
\min(m, n) \cdot (\log_2 n)^2 \\
&\leq m \cdot (\log_2 (am^b))^2 \\
&= m(\log_2 a + b\log_2 m)^2 \\
&= m(\log_2 a)^2 + 2b\log_2 a \cdot m\log_2 m + b^2 m(\log_2 m)^2 \\
&\leq 2^I (\log_2 a)^2 + 2b\log_2 a \cdot 2^I \cdot \log_2 2^I + b^2 2^I (\log_2 2^I)^2 \\
&= 2^I (\log_2 a)^2 + b\log_2 a \cdot 2^I \cdot I + b^2 2^I (\frac{I}{2})^2 \\
&= O(2^I I^2)
\end{align*}
\]

- 3 points for showing $\min(m, n) \cdot (\log_2 n)^2$.
- 4 points for showing calculation.
- 2 points for giving the final result $O(2^I I^2)$. 

12
8. (15 Points)
Show that the Bin Packing (BP) problem is NP-complete. You may use the fact that the Set Partition (SP) problem is NP-complete. Here are the problem definitions.

**Bin Packing (BP):** The input is a finite set \( S' \) of integers, and two other integers \( b \) and \( k \). The problem is to determine whether the set can be partitioned into \( k \) subsets such that the sum of the numbers in each subset is at most \( b \).

**Set Partition (SP):** The input is a finite set \( S \) of positive integers. The problem is to determine whether the set \( S \) can be partitioned into two subsets such that the sum of the numbers in each subset is the same.

**Solution:**

1. Show Bin Packing (BP) is in Class NP. [5 points]
   
   (a) Give a certificate which corresponds to a yes-input (\( \uparrow 2 \) points)
   
   The certificate consists of partition of \( S'' \), denoted as \( \{S''_1, \ldots, S''_m\} \).

   (b) Describe a verification algorithm to show that the input is indeed a yes-input (\( \downarrow 2 \) points)
   
   Given the partition of \( S'' \), we check whether the sum of each \( S''_i \) is less than or equal to \( b \). i.e. \( \sum_{x \in S''_i} x \leq b \) for each \( i \). Also, we need to check whether \( m = k \). If both of them are true, then the algorithm outputs yes, otherwise outputs no.

   (c) Analyze the running time of the verification algorithm (show that it is polynomial) (\( \downarrow 1 \) point)
   
   Input-size is \( s = \sum_{x \in S''} \log_2 x + \log_2 b + \log_2 k \).
   
   Verification can be done in time
   \[
   O\left(\sum_{i=1}^{m} \left( \sum_{x \in S''_i} \log_2 x + \log_2 b + \log_2 k \right) \right) = O(s)
   \]
   
   which is polynomial with respect to the actual input size \( s \).

2. Choose a known NPC decision problem (we will use SP) and REDUCE IT to BP in polynomial time. (i.e. SP \( \leq_P \) BP) [10 points]

   (a) General description of the polynomial-time reduction (transformation) \( f \) (\( \downarrow 2 \) points)
   
   (i.e. Showing the input relationship between SP and BP in terms of the context.)
   
   We want a polynomial computable function \( f \), which given an instance of SP (a set of positive integers \( S \)) outputs an instance of BP (a set of positive integers \( S' \), and two other integers \( b \) and \( k \)) such that the set \( S \) can be partitioned into two subsets in which the sum of the numbers in each subset is the same IF AND ONLY IF the set \( S' \) can be partitioned into \( k \) subsets and the sum of the numbers in each subset is at most \( b \).
(b) Details of the reduction (transformation) algorithm \( f \) ( \( \downarrow \) 3 points)

We are given a set partition instance \( S = \{s_1, s_2, \ldots, s_n\} \).

We will create \( S' \) as the same set as \( S \). And we calculate the total sum of all positive integers in \( S \), denoted as \( S_{\text{total}} = \sum_{i=1}^{n} s_i \) and set \( b = \left\lfloor \frac{1}{2} S_{\text{total}} \right\rfloor \). Also we assign \( k = 2 \).

Then we output \( S', b \) and \( k \) as the instance of the bin-packing problem.

(c) Running time of the reduction (transformation) ( \( \downarrow \) 1 point)

(i.e. Showing that the transformation runs in polynomial time.)

The duplication of set \( S \) takes \( O(\sum_{x \in S} \log_2 x) \). For the calculation of \( S_{\text{total}} \), it takes \( O(\sum_{x \in S} \log_2 x) \) time. And the assignment of \( b \) and \( k \) take \( O(\log_2 S_{\text{total}}) \).

So, the reduction takes \( O(\sum_{x \in S} \log_2 x + \log_2 S_{\text{total}}) \) time which is polynomial with respect to the input size \( s = \sum_{x \in S} \log_2 x \).

(d) Correctness of the reduction (transformation) ( \( \downarrow \) 4 points)

Suppose the set-partition instance is a yes-instance, then there exist two partitions \( S_1 \) and \( S_2 = S - S_1 \) which sum to the same value which is equal to \( S_{\text{total}}/2 \). Thus, the set \( S' = S \) can be partitioned into 2 subsets such that the sum of the numbers in each subset is at most \( S_{\text{total}}/2 \).

Suppose the set-partition instance is a no-instance, then there doesn’t exist two partitions which sum to the same value. This implies the sum of items in one set will exceed \( \left\lfloor S_{\text{total}}/2 \right\rfloor \), and thus the set \( S' = S \) need to be partitioned into 3 subsets such that the sum of the number in each subset is at most \( \left\lfloor S_{\text{total}}/2 \right\rfloor \).

Note: If students were able to give correct transformation, running time and correctness proof, full marks should be given even they didn’t write down the general description.